International Journal of Computer Science Engineering
and Information Technology Research (IJCSEITR)
ISSN(P): 2249-6831; ISSN(E): 2249-7943
Vol. 5, Issue 4, Aug 2015, 63-70

TRANS STELLAR
• Journal Publications • Research Consultancy

INDEX BASED DETECTION OF FILE LEVEL CLONE FOR HIGH LEVEL CLONING

MANU SINGH & VIDUSHI SHARMA

School of ICT, Gautam Buddha University, Yamuna Expressway, Greater Noida, Uttr Pradesh, India

ABSTRACT

In modern time cloning at high level of abstraction is a promising idea that organize the simple level clone and form the bigger level clone. High level of cloning aggregates the similarity at coarser level in a system. These high level clones are simple clones, clones with same behaviour, clones with same model, clones with similar concept and clones with similar structure. The Structural clones are formed by lower level smaller clones with similar code fragments. In this recurrent occurrence of simple clones in a file may lead to higher file level clones. The proposed algorithm detects high level clones in terms of file clones and also detects redundancy in patterns occurrence in same file. One of the promising research areas in computer science is the study of pattern matching. There are various pattern matching approaches used in biological problems to identify DNA sequences. We propose an algorithm based on Index based Pattern Matching Algorithms using frequent character count in patterns. Algorithm proposed depicts index based detection of file level cloning which reduces the time complexity of algorithm.

KEYWORDS: Pattern Matching, Index Based, High Level Clone, File Clone

INTRODUCTION

Due to change in technological and functional features, the software system required time to time upgrading and maintenance. Due to modifications in code, redundancy occur in code and software will be more complex and difficult in maintaining. Sometimes this redundancy is known as cloning. Cloning occurs at various abstraction levels and may possibly have different source [1]. Literature study depicts 50% cloning in the source code [2]. Several proposed approaches used to identify clone fragments which are similar to each other these clone fragments are known as simple clones [3] but similarity analysis at higher levels remains a promising area till now. The research of pattern matching approaches is one of the promising areas of research in cloning. Pattern matching decreases the number of comparisons and time complexity of algorithm, which are required in the worst case and average case analysis. This paper investigate the applicability of a new technique of pattern matching approach called Index based Pattern Matching algorithm, for detection of high level clone in source code. This approach avoids lengthy comparisons in string sequence and reduces the effort for each character comparison at each attempt. The proposed algorithm gives better results as compared to other algorithms.

High Level Clones are classified [4] in structural clone, concept clone, behavioural clone [5] and domain model clone. This classification depicts that structural clones are formed by similar fragments of code at low level. In this recurrent occurrence of simple clones in a file may lead to higher file level clones. The proposed algorithm detects high level clones in terms of file clones.

Related Work

There are various string matching techniques which mainly deal with problem of identifying occurrences of a substring in a given string or locate the occurrences of specific pattern in a sequence. In this section we explore these

different types of string matching techniques. Some techniques are based on algorithms of exact matching in string, such as Naïve string search, Brute-force algorithm, Bayer-Moore algorithm, Knuth-Morris-Pratt algorithms [6],[7] and some are based on approximate string matching algorithms, dynamic programming is mostly used approach. In identifying exact pattern matching Needleman [9] algorithm and waterman algorithms are complex. The complexity of these approaches is O(MN).

Brute-force algorithm is a straight forward approach in which 1st character of pattern is compare with the 1st character of the text. If match is found at first position then pattern and text are compared with each other character by character. Once the match is found or difference occur at some position the text comparison continues.. If difference occurs, the pattern is aligned to the next character. The Worst case time complexity of Brute-force algorithm is O(MN).

The Bayer-Moore algorithm [6] compares group of characters and then shifting the pattern to the right is done. The main aspect of the algorithm is to match on the pattern left side rather than the right, and to skip multiple characters rather than searching each and every character in the text. i.e., after aligning Pattern and text the end character of Pattern will be matched to Text first. If a difference is identified, say character in Text is not match with character in Pattern then Pattern is shift to the right to align the character in text with the extremely right occurrence of character in Pattern. The complexity of Bayer-Moore algorithm is O(M+N). Another algorithm is Knuth-Morris-Pratt algorithm [7], which is based upon finite state automation machine. The pattern is converted to a finite state automation machine. The finite state automation machine is generally represented by the transition table. The Worst case time complexity of the algorithm is O(M+N). Kurtz [8] proposed an approach to decrease the requirement of space and build only the actually required states and transitions in the text processing. The automation begins and transitions are built as they are required. The transitions those were not required will not be build in the state automation machine.

Ukkonen [10] proposed an automation based approach for finding approximate patterns in texts. For identifying some expression of approximate pattern matching, Wu.S.Manber and Myer.E [11] proposed the algorithm. For solving the problem of inexact pattern matching Wu.S.Manber.U [12] proposed an approach using a deterministic finite automaton. According to exact pattern matching there is no big advantage of time over Boyer-Moore algorithm[6] The O(M+N) is the time complexity of Boyer-Moore algorithm algorithm. Deterministic automata approach indicates O(N) worst case complexity of time. The main limitation of this approach is construction of the deterministic finite automaton from nondeterministic finite automaton which takes exponential time and space. In the Index based forward backward pattern matching approch[13] the characters in the given patterns are matched sequentially in the forward and backward direction until a difference in character or complete match occurred. This paper proposed the most efficient approach for finding similarity between multiple pattern, till date. It is a simple approach for finding multiple occurrences of patterns from a given file. This algorithm gives better results when compare it with existing algorithms. This approach provides best results with the DNA sequence dataset.

Algorithm

Input: S String with n words and P pattern with m words, where S, P belongs to source code of a program.

Output: The total occurrences of P Pattern in S String, its position and the total number of words compared.

Step 1: [Initialization of variables & Index array] Integer

Count:=0 and ncmp:=0, Indexarr[n]

Step 2: In pattern P the frequent repeatedly occurred word are identified.

Freqword=patternarr[]

Step 3: The frequent word of P Pattern is compare with the S string and the matching index is written to the index table Indexarr[];

For (a=1 to stringLength)

If string(a)!=freqword

Continue;

else

Indextable(b)=a;

b=b+1

endif

End

Step 4: Based on the value of index aligned the P pattern with S string.

Start=indextable(a) – patternindex

End = start + patternLength – 1;

If end>substringLength

Break;

Endif

Outstr=substr(start:end)

If outstrLength < patternLength

Exit

End

Step 5: compare the ascii sum of string and ascii sum of pattern.

If substr_asc == pattern_asc

Step 6: If they are equal compare individual words of string and pattern.

For i=1 to patternLength

Num=num+1

If (outstr(i) == pattern(i))

Found =1;

continue;

else

Break;

endif

End

Step 7 : if found==1

ctr=ctr+1

print (found at location, start)

endif

End

Working Example

Assume there is a string to understand the proposed algorithm

S= RED GREEN BLUE GREEN RED WHITE BLUE BLUE BLUE GREEN RED WHITE BLUE RED RED GREEN BLUE GREEN RED WHITE WHITE BLUE GREEN RED WHITE BLUE GREEN BLUE WHITE of 29 words

P = BLUE GREEN RED WHITE BLUE of 5 words

The occurrence of frequent repeatedly word of Pattern P is identified. In this example the frequent word of pattern is BLUE.

Now based on the frequently occurred word of Pattern, find the pattern index of that word in the String S and write to an index table. Table 1 shows the index table for character BLUE of S String.

Table 1: Frequent Word Index Table

3	**7**	**8**	**9**	**13**	**17**	**22**	**26**	**28**

S= RED GREEN BLUE GREEN RED WHITE BLUE BLUE BLUE GREEN RED WHITE BLUE RED RED GREEN BLUE GREEN RED WHITE WHITE BLUE GREEN RED WHITE BLUE GREEN BLUE WHITE

P = BLUE GREEN RED WHITE BLUE

The first word matches then it compares the second word of pattern with next word in string.

S= RED GREEN BLUE GREEN RED WHITE BLUE BLUE BLUE GREEN RED WHITE BLUE RED RED GREEN BLUE GREEN RED WHITE WHITE BLUE GREEN RED WHITE BLUE GREEN BLUE WHITE

P = BLUE GREEN RED WHITE BLUE

The second word also matches then it compares the third word.

S= RED GREEN BLUE GREEN RED WHITE BLUE BLUE BLUE GREEN RED WHITE BLUE RED RED GREEN BLUE GREEN RED WHITE WHITE BLUE GREEN RED WHITE BLUE GREEN BLUE WHITE

P = BLUE GREEN RED WHITE BLUE

S= RED GREEN BLUE GREEN RED WHITE BLUE BLUE BLUE GREEN RED WHITE BLUE RED RED GREEN BLUE GREEN RED WHITE WHITE BLUE GREEN RED WHITE BLUE GREEN BLUE WHITE

P = BLUE GREEN RED WHITE BLUE

All the word are matched, so the pattern is found at position 3.

Next the pattern is aligned with string based on the second index of the index table.

S= RED GREEN BLUE GREEN RED WHITE BLUE BLUE BLUE GREEN RED WHITE BLUE RED RED GREEN BLUE GREEN RED WHITE WHITE BLUE GREEN RED WHITE BLUE GREEN BLUE WHITE

P= BLUE GREEN RED WHITE BLUE

Ascii sum of Pattern and Ascii sum of substring are not match with each other. Next align the pattern with second index of the index table.

S= RED GREEN BLUE GREEN RED WHITE BLUE BLUE BLUE GREEN RED WHITE BLUE RED RED GREEN BLUE GREEN RED WHITE WHITE BLUE GREEN RED WHITE BLUE GREEN BLUE WHITE

P= BLUE GREEN RED WHITE BLUE

Ascii sum of Pattern and Ascii sum of substring matched. Now compare the pattern with substring word by word. After comparing the pattern was not found at that position. So it is a false positive result. Now aligned with the fourth index

S= RED GREEN BLUE GREEN RED WHITE BLUE BLUE BLUE GREEN RED WHITE BLUE RED RED GREEN BLUE GREEN RED WHITE WHITE BLUE GREEN RED WHITE BLUE GREEN BLUE WHITE

P= BLUE GREEN RED WHITE BLUE

Here Ascii sum of Pattern and Ascii sum of substring are matched Now compare the pattern with substring word by word.

BLUE GREEN RED WHITE BLUE

BLUE GREEN RED WHITE BLUE

All the word are matched, so the pattern is found at position 9. Now pattern is aligned with String according to the fifth index of the index table and it compares Ascii sum of Pattern and Ascii sum of substring.

S= RED GREEN BLUE GREEN RED WHITE BLUE BLUE BLUE GREEN RED WHITE BLUE RED RED GREEN BLUE GREEN RED WHITE WHITE BLUE GREEN RED WHITE BLUE GREEN BLUE WHITE

P= BLUE GREEN RED WHITE BLUE

Ascii sum of Pattern and Ascii sum of substring are not match with each other. Now the pattern is aligned based on the next index

S= RED GREEN BLUE GREEN RED WHITE BLUE BLUE BLUE GREEN RED WHITE BLUE RED RED GREEN BLUE GREEN RED WHITE WHITE BLUE GREEN RED WHITE BLUE GREEN BLUE WHITE

P=BLUE GREEN RED WHITE BLUE

The values does not match and the pattern is aligned based on the next index

S= RED GREEN BLUE GREEN RED WHITE BLUE BLUE BLUE GREEN RED WHITE BLUE RED RED GREEN BLUE GREEN RED WHITE WHITE BLUE GREEN RED WHITE BLUE GREEN BLUE WHITE

P= BLUE GREEN RED WHITE BLUE

All the word are matched, so the pattern is found at position 22.

Now the pattern is moved to the next index and the comparison does not occurs and algorithm terminates because the length of substring is less than the pattern length. Now at the end of the algorithm the total 3 Patterns occurred in the String. By above example we can conclude that taking frequent count reduces the number of comparisons.

Comparative Analysis

The proposed algorithm is compared with existing algorithms. Table 2 gives the summary of some existing and proposed algorithm.

Table 2: Summary of String Matching Algorithm

Name of the Algorithm	Order of Comparison	Processing Required	Time Complexity in Searching
Brute-force	Not applicable	No	O(mn)
Bayer-Moore[6]	Right to left	Yes	O(mn)
Knuth-Morris-Pratt[7]	Left to Right	Yes	O(m+n)
Index Based	Left to Right	Yes	O(mn)

It can be analyzed that but the salient features of index based algorithm gives an improvement to other algorithms are following:

- Decreases number of comparisons in average and best case analysis
- Appropriate for very large size input file
- Indexes are created once

CONCLUSIONS

A new algorithm Index Based Detection Of File Level Clone for High Level Cloning is proposed. This paper gives the time efficient method for solving pattern matching problem. This approach is used to identify occurrence of patterns with multiple times in a file. In future the proposed algorithm can be tested with real time or large dataset to estimate its performance.

REFERENCES

1. H. A. Basit, S. Jarzabek, " A Case for Structural Clones ", International Workshop on Software Clones , 2009.
2. B. S Baker, "On Finding Duplication and Near duplication in Large Software System ", Proceedings of 2nd IEEE Conference of Reverse Engineering , 1995.
3. William S. Evans , Christopher W. Fraser and Fei Ma , " Clone Detection via Structural Abstraction ", Software quality journal Vol. 17, No. 4, 2009.
4. M. Singh, V. Sharma, "High Level Clones Classification" International Journal of Engineering and Advanced Technology (IJEAT) ISSN : 2249 – 8958, Vol. 2, Issue - 6, August 2013.
5. M. Singh, V. Sharma, "Detection of Behavioral Clone International Journal of Computer Applications (0975 – 8887) Vol. 102 – No.14 , 2014.
6. Bayer R. S., J. S. Moore, "A Fast String Searching Algorithm " , Communications of the ACM , pp. 762-772 ,1977
7. Knuth D., Morris.J ,Pratt.V.R., "Fast Pattern Matching in Strings ", SIAM Journal on Computing Vol. 6 (1), 1977.
8. Kurtz. S, "Approximate string searching under weighted edit distance", In proceedings of the 3 South American workshop on string processing. (WSP 96). Carleton Univ Press, 1996 pp. 156-170.
9. Needleman , S.B Wunsch, C .D.,"A general method applicable to the search for similarities in the amino acid sequence of two proteins", J. Mol. Biol., Vol. 48 , pp. 443-453,1970.
10. Ukkonen,E., "Finding approximate patterns in strings", J. Algor. ,Vol 6, pp. 132-137, 1985.
11. WU.S.,Manber U., and Myers,E .1996, A sub-quadratic algorithm for approximate limited expression matching. Algorithmica 15,1,50-67, Computer Science Dept, University of Arizona,1992
12. Wu S., and U. Manber, ''Agrep — A Fast Approximate Pattern-Matching Tool,'' Usenix Winter 1992 Technical Conference, San Francisco (January 1992), pp. 153 162.
13. Raju Bhukya, DVLN Somayajulu,..An Index Based Forward backward Multiple Pattern Matching Algorithm,. World Academy of Science and Technology. (June 2010), pp. 347-355.